



Java Fundamentals

3-6

Defining Methods

```
private void catchFly() {  
    if (isTouching(Fly.class)) {  
        removeTouching(Fly.class);  
    }  
}
```

Objectives

This lesson covers the following objectives:

- Describe effective placement of methods in a super or subclass
- Simplify programming by creating and calling defined methods
- Handling collisions



Efficient Placement of Methods

- At times, many lines of code are required to program a behavior.
 - For example, you may want to program an instance to eat other objects, or turn when it reaches the edge of the world.
- You can define new methods to simplify logic, save time, and use fewer lines of code
 - Define (code) a new method for an action below the act method
 - Call (use) the new method in the act method of a world or actor, or within another method
 - Define (code) the method in the superclass if you want its subclasses to automatically inherit the method



Defined Methods

- Defined methods are new methods created by the programmer.
- These methods:
 - Can be executed immediately, or stored and called later.
 - Do not change the behavior of the class when stored.
 - Separate code into shorter methods, making it easier to read.

Defined methods create a new method that a class did not already possess. These methods are written in a class's source code below the act method.



Turn at the Edge of the World

- Problem:

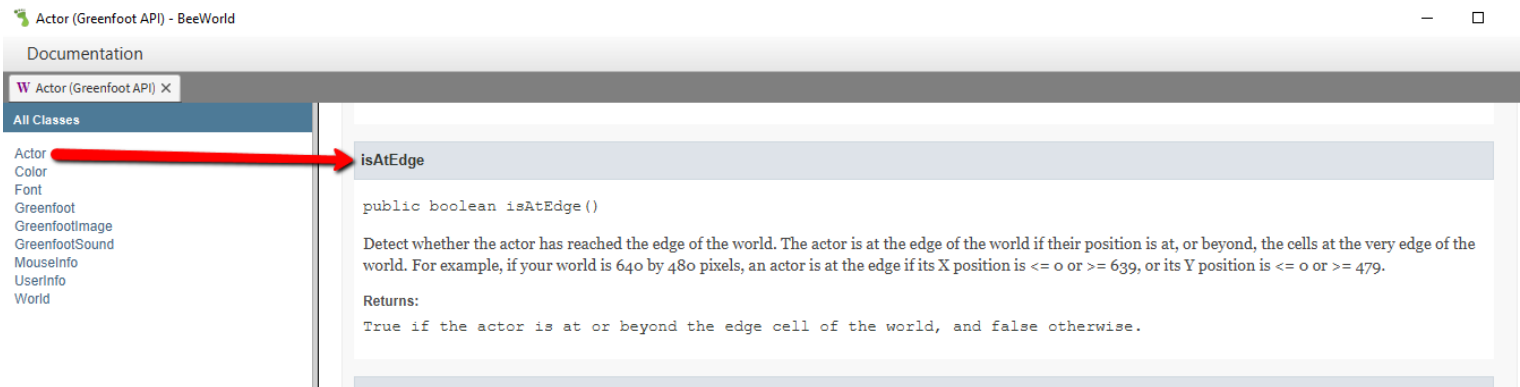
- Instances stop and are unable to move when they reach the edge of the world.
- Instances should turn and move in the opposite direction when they reach the edge of the world.

- Solution:

- Create a subclass of Actor that defines a method that can detect if the object is at the edge of the world and to turn appropriately.
- Create subclasses of the Actor subclass that should inherit the method.
- Call (use) the new method in the subclasses that should be able to turn and move at the edge of the world.

Test if an Actor is at the edge of the World

- Greenfoot has a method in the Actor class called `isAtEdge()`.
- This returns true if the Actor is at one of the edges.
- We can use this to detect and then turn actors around rather than them hover at one of the edges.



Test an Object's Position in the World

To test if an object is near the edge of the world:

- Use an **if** statement with the boolean `isAtEdge()` method to test if the condition is true or false.
- Example: We can rotate an instance by 180 degrees if its at the edge of the world.

```
public void act()  
{  
    move(1);  
    if (isAtEdge()) {  
        turn(180);  
    }  
}
```


Logic Operators

Logic operators can be added to if/else statements to combine multiple boolean expressions into one boolean expression.

Logic Operator	Means	Definition
Exclamation Mark (!)	NOT	Reverses the value of a boolean expression (if b is true, !b is false. If b is false, !b is true).
Double ampersand (&&)	AND	Combines two boolean values, and returns a boolean value which is true if and only if both of its operands are true.
Two lines ()	OR	Combines two boolean variables or expressions and returns a result that is true if either or both of its operands are true.

Logic Operators - Examples

- If a Spider is in the top right half of the scenario, move the Spider forward:

```
public void act()  
{  
    if((getX() > getWorld().getWidth()/2) && getY() < getWorld().getHeight()/2)  
        move(2);  
    //endif  
}
```

- If the Actor's horizontal position is greater than the center of the horizontal world, and the Actor's vertical position is less than the center of the vertical world...
 - getX() returns the horizontal position of the Actor
 - getWorld().getWidth() returns the horizontal size of the World
 - getY() returns the vertical position of the Actor
 - getWorld().getHeight() returns the vertical size of the World

Logic Operators - Examples

Substitute `||` for the `&&` to test if a Spider is in the top half OR the right half of the scenario.

```
public void act()  
{  
    if((getX() > getWorld().getWidth()/2) || getY() < getWorld().getHeight()/2)  
        move(2);  
    //endif  
}
```

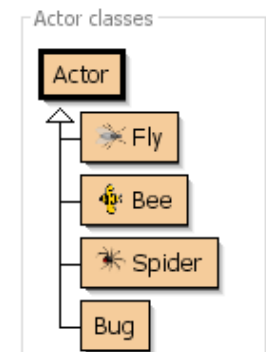
Logic Operators - Examples

Use ! to move a Spider forward if it is NOT in the right half of the scenario.

```
public void act()  
{  
    if(!(getX() > getWorld().getWidth()/2 || getY() < getWorld().getHeight()/2)  
        move(2);  
    //endif  
}
```

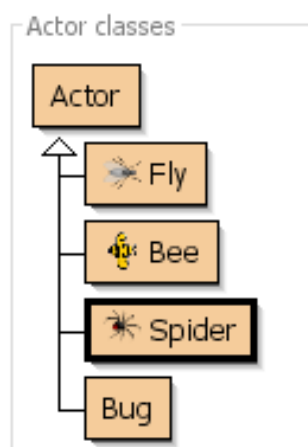
Creating a Superclass

- To allow both the Spider and the Fly classes to inherit the turning behavior we place the code in a superclass.
- The name of this superclass should be descriptive of the commonality of the classes that will inherit from it.
- We will create a Bug class that has no image and will not have instances that act in the scenario, but will hold defined methods that its subclasses will inherit.
 - Right+click Actor, choose New subclass
 - Name the new class “Bug”
 - Do not assign an image
 - Click Ok

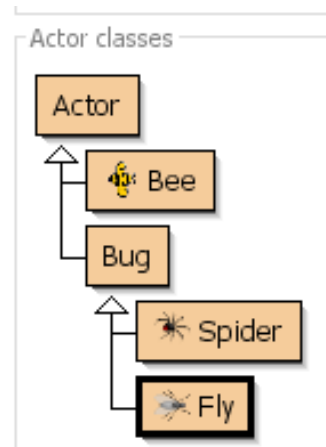


Create the Bug Subclasses

- We could recreate our Spider and Fly by right clicking on Bug, and selecting new Subclass.
- But as we have previously created them we can modify our Spider and Fly source code to extend from Bug rather than Actor.



```
public class Spider extends Bug
{
    public void act()
    {
        // Add your action code here.
    }
}
```



Define turnAtEdge() Method in Superclass

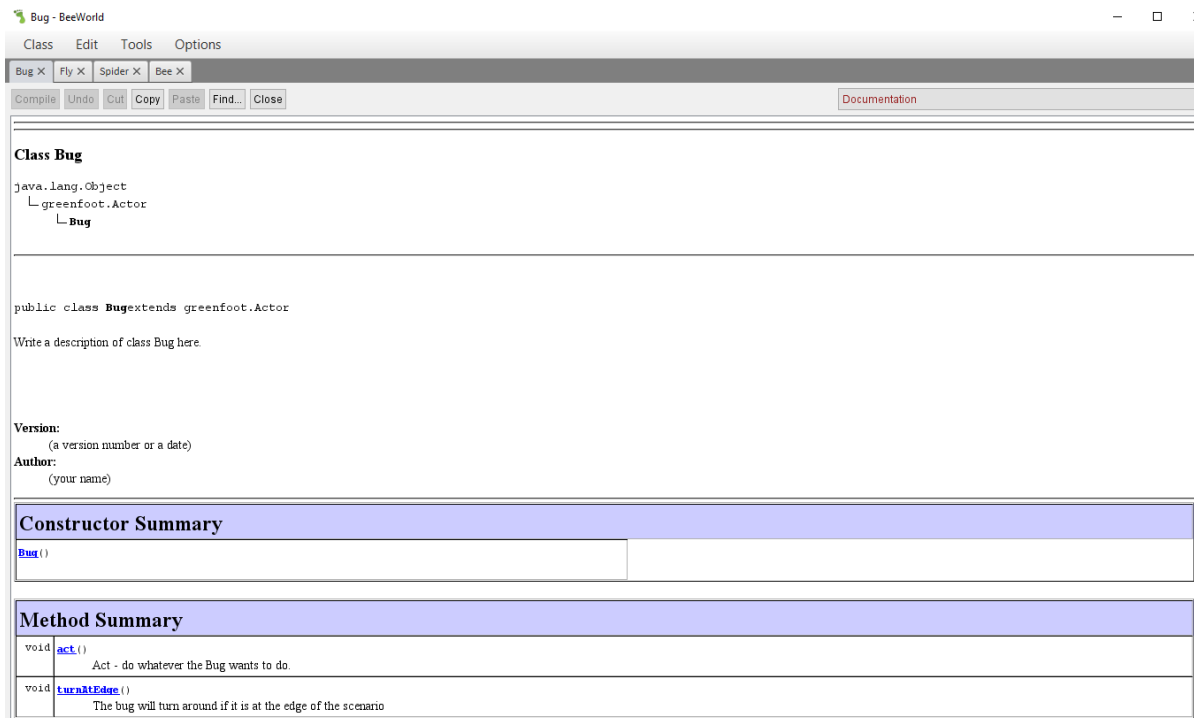
Open the Code editor for the Bug class. Write the code for the turnAtEdge() method, below the act() method.

```
public class Bug extends Actor
{
    /**
     * Act - do whatever the Bug wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        // Add your action code here.
    }

    /**
     * turnAtEdge - turn the actor 180degrees if at the edge of the world.
     */
    public void turnAtEdge()
    {
        if(isAtEdge())
            turn(180);
        //endif
    }
}
```

Class Documentation

The Bug class documentation shows the new method after its defined.



Call turnAtEdge() Method in Subclass

- Open the Code editor for the Fly subclass from the previous lesson.
- Add a call to the method turnAtEdge() within the act() method.
- Repeat this for the Spider subclass.

```
public void act()
{
    move(1);
    if (Greenfoot.getRandomNumber(100) < 10) {
        turn(Greenfoot.getRandomNumber(90)-45);
    }
    turnAtEdge();
}
```

Define at[left, right, top, bottom]Edge() Methods in Bee class

- For the Bee class, we will write methods that will
 - determine the edge where the Bee is approaching
 - move the Bee to the opposite edge (appears to wrap around)
- For example, if the Bee is moving toward the right, and approaching the right edge:
 - it will disappear from the right edge
 - re-appear on the left edge
 - continue moving toward the right edge again
- To know which edge an actor is touching, we will define 4 separate methods, one for each side.

Define atRightEdge() Method in Bee class


- Open the Code editor for the Bee class.
- Write the code for the atRightEdge() method, below the act method.

```
/**
 * Test if we are close to the right edge of the world. Return true if the object is.
 */
public boolean atRightEdge()
{
    if(getX() > getWorld().getWidth() - 20 )
        return true;
    else
        return false;
}
```

Define atBottomEdge() Method in Bee Class

- Open the Code editor for the Bee class.
- Write the code for the atBottomEdge() method, below the act method.

```
/**
 * Test if we are close to the bottom edge of the world. Return true if the object is.
 */
public boolean atBottomEdge()
{
    if(getY() > getWorld().getHeight() - 20 )
        return true;
    else
        return false;
}
```



Methods called from `atRightEdge()` and `atBottomEdge()`

The methods used in `atRightEdge()` include:

- `getX()`: An Actor method that returns the x-coordinate of the actor's current location.
- `getY()`: An Actor method that returns the y-coordinate of the actor's current location.
- `getWorld()`: An Actor method that returns the world that this actor lives in.
- `getHeight()`: A World class method that returns the height of the world.
- `getWidth()`: A World class method that returns the width of the world.

Call Methods in Bee Class

- Open the Code editor for the Bee class.
- Create an IF statement that calls the `atRightEdge()` and `atBottomEdge()` method as a condition in `act`.
- If the Bee is at the left it will re-appear on the right and vice versa.

Move will always execute.

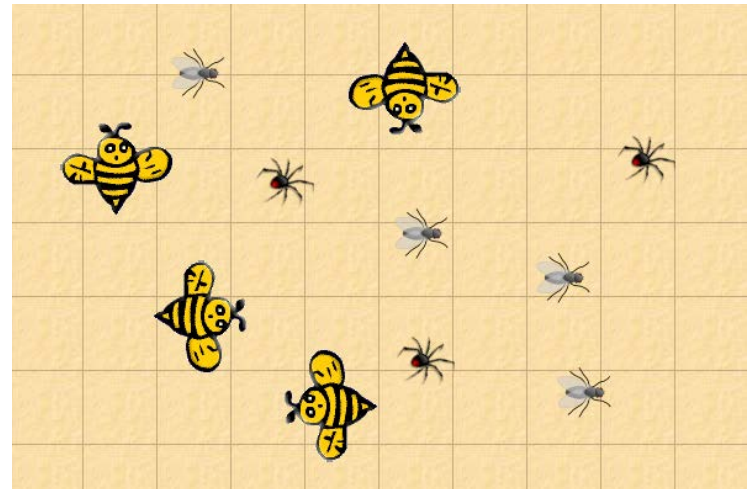
If the Bee is near the right edge, the x co-ordinate will be set to 6. The current Y position is kept using the `getY()` method.

If the Bee is near the bottom edge, the Y co-ordinate will be set to 6. The current X position is kept using the `getX()` method.

```
public void act()
{
    move(1);
    if(atRightEdge()) {
        setLocation(6, getY());
    }else
    if(atBottomEdge()) {
        setLocation(getX(), 6);
    }//endif
}
```

Call Methods in Class

- Open the Code editor for the myWorld class
- If necessary,
 - add four Bee objects to your world
 - select Controls -> Save the World
- In the prepare() method, add the following:
 - `bee2.turn(90);`
 - `bee3.turn(-90);`
 - `bee4.turn(180);`



Call Method in Class

- Complete the IF statement for the `atLeftEdge()` and `atTopEdge()`.
- Test your program to make sure the Bee instances move off the world edges to the opposite edge

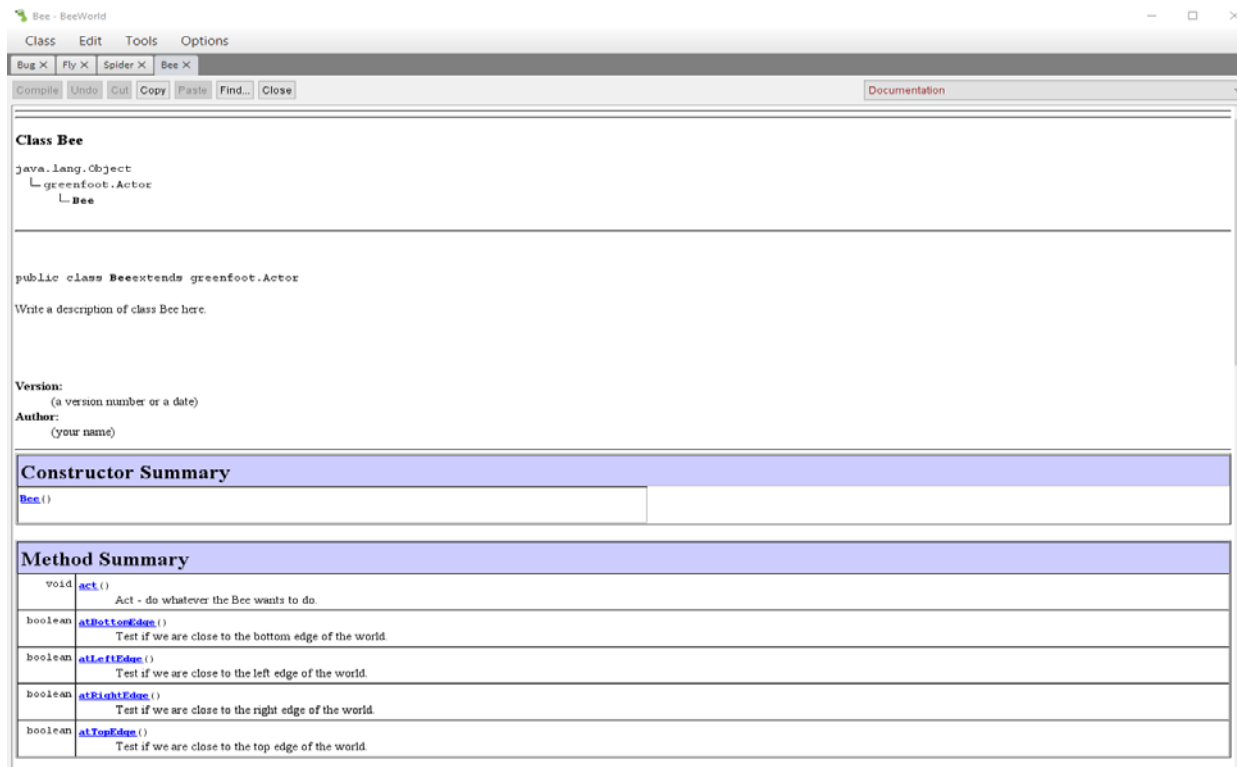
```
public boolean atLeftEdge()
{
    if(getX() < 6 )
        return true;
    else
        return false;
}
```

```
public boolean atTopEdge()
{
    if(getY() < 6 )
        return true;
    else
        return false;
}
```

```
public void act()
{
    move(1);
    if(atRightEdge()){
        setLocation(6, getY());
    }
    else if(atBottomEdge()){
        setLocation(getX(), 6);
    }
    else if(atLeftEdge()){
        setLocation(getWorld().getWidth()- 6, getY());
    }
    else if(atTopEdge()){
        setLocation(getX(), getWorld().getHeight() - 6);
    }
}
```


Class Documentation

The Bee class documentation shows the new method after its defined.



Collisions

- Most game projects will have to detect when two actors touch, which is called a collision.
- In GreenFoot there are multiple ways to detect this.
- Some of these are:
 - `isTouching()`
 - `getOneIntersectingObject(Class)`
 - `getOneObjectAtOffset(Class)`
 - `getIntersectingObjects(Class)`
 - `getNeighbours(distance,diagonal)`
 - `getObjectsAtOffset(dx,dy,Class)`

Collisions

Method	When To Use
<code>isTouching()</code>	When you want to detect a collision with an object
<code>getOneIntersectingObject()</code>	When you want to return a reference to the object you have collided with. Use to perform an action on the collided object.
<code>getOneObjectAtOffset()</code>	Same as <code>getOneIntersectingObject()</code> , except that you can change where the collision will be detected relative to the current object. So you could have the collision detected before it happens. i.e., to stop an actor walking into a wall.

Defined Method to Remove Objects

- You can write code in your game so a predator object is able to eat prey objects.
- Create a defined method in the Bee class called `catchfly()` to enable us to remove the flies that the Bee catches.
- To create this defined method we are going to use the simplest collision detection – `isTouching()`.
- This method detects a Bee's collision with a Fly, and then removes it.

```
private void catchFly() {  
    if (isTouching(Fly.class)) {  
        removeTouching(Fly.class);  
    }  
}
```

Define catchfly() Method - Alternative

Alternatively we could have used `getOneIntersectingObject()`, and accessed a reference to the actor before deleting it.

```
private void catchFly2() {  
    Actor fly = getOneIntersectingObject(Fly.class);  
    if (fly != null) {  
        getWorld().removeObject(fly);  
    }  
}
```

Call catchfly() in act() Method

- Call the new catchfly() method in the Bee's act() method.
- Be careful to add the code outside the if statement as we want the bee to always try to catch a fly.
- Run the scenario to test the code.

```
public void act()
{
    move(1);
    if(atRightEdge()){
        setLocation(6, getY());
    }else
    if(atBottomEdge()){
        setLocation(getX(), 6);
    }else
    if(atLeftEdge()){
        setLocation(getWorld().getWidth()-6, getY());
    }else
    if(atTopEdge()){
        setLocation(getX(), getWorld().getHeight()-6);
    }//endif
    catchFly();
}
```

Call catchfly() in act() Method

Additional changes recommended for the act() method:

- re-locate the move() statement from the act() method to a new method, called handleMovement()
- re-locate the if statements from the act() method that check if the Bee is at the edge by adding a turnAtEdge() method that tests if the Bee is at the edge

```
public void turnAtEdge(){  
    if(atRightEdge()){  
        setLocation(6, getY());  
    }  
    else if(atBottomEdge()){  
        setLocation(getX(), 6);  
    }  
    else if(atLeftEdge()){  
        setLocation(getWorld().getWidth() - 6, getY());  
    }  
    else if(atTopEdge()){  
        setLocation(getX(), getWorld().getHeight() - 6);  
    }  
}
```

```
public void handleMovement()  
{  
    move(1);  
}
```

```
public void act()  
{  
    handleMovement();  
    turnAtEdge();  
    catchFly();  
}
```

Terminology

Key terms used in this lesson included:

- Defined methods
- Collisions

Summary

In this lesson, you should have learned how to:

- Describe effective placement of methods in a super or subclass
- Simplify programming by creating and calling defined methods

